اساسيات البرمجة الموجهة بالهدف باستخدام لغة الجافا

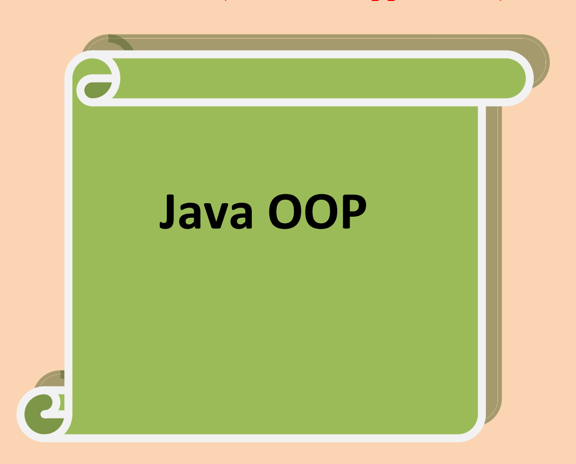
OOP Fundamental Using Java

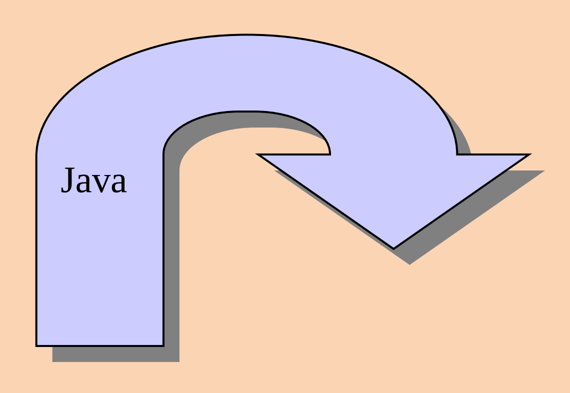
By: Salem.M.Adrugi

Java

Programming Language

(Console Application)





المحاضر : **سالم مسعود الدروقي**

جامعة المرقب-كلية التربية -الخمس

2016



NetBeans IDE

My NetBeans

Recent Projects

<no recent project>

ORACLE



البرمجة الموجهة بالهدف Object Oriented Programming:

البرمجة الموجهة بالهدف (Object Oriented Programming) وتختصر (OOP) هي عبارة اسلوب برمجة يعتمد اساساً على محاكاة نمط الحياة الحقيقة (Real Live) من خلال التركيز على مفهوم فئات الكائنات والبيانات بدلاً من الاحداث والمنطق، حيث يعتمد اسلوب البرمجة الموجهة بالهدف على تقسيم البرنامج إلى فئات (Classes) تضم كل فئة مجموعة من الكائنات (Objects) المتشابهة في الخصاص والافعال، وعند النظر إلى واقع الحياة الحقيقية نجد أن كل ما هو موجود في هذه الحياة من كائنات حية وجماد ما هو إلا كائن له مجموعة من الخصائص والافعال ويندرج هذا الكائن تحت فئة او فصيلة تضم مجموعة من الكائنات المتشابه معه في العديد من الصفات والافعال.

مثال1: **الفئة مركبة:** يمكن أن نشتق منها الكائنات (سيارة، دراجة، طائرة، شاحنة).
والتي يمكن أن تحتوي كل منها على مجموعة من الخصائص مثل (النوع، اللون، قوة المحرك، عدد الركاب، تاريخ الصنع).

كما تستطيع هذه الكائنات القيام ببعض الافعال المشتركة مثل (الانطلاق ، التوقف ، تشغيل المحرك ، ايقاف تشغيل المحرك)

مثال2:

الفئة طالب: يمكن أن نشتق منها الكائنات (Ahmed,Salem,Mhamed)

والتي يمكن أن تحتوي كل منها على مجموعة من الخصائص مثال (الاسم ، رقم القيد ، السنة الدراسية ، المعدل الفصلي).

كما تستطيع هذه الكائنات القيام ببعض الافعال المشتركة مثل (الكتابة على الورق ،حضور محاضرة ، الاستماع للمحاضرة ، دخول الامتحان).

وفيما يلي نقدم اهم المفاهيم الرئيسة التي تعتمد عليها اساليب البرمجة الموجهة بالهدف.

الفئات و الكائنات Classes & Objects :

الفئة (Class): هي تمثيل شامل لنوع معين من الأشياء نستطيع من خلاله اشتقاق مجموعة كائنات تتشابه في العديد من الخصائص بمعني أن الفئة عبارة عن قالب عام نستطيع من خلاله اشتقاق الكائنات ويمكن تمثيل الفئة بالشكل التالي:

Class_Name
Class_Properties
Class_Methods

مثال:

Circle
-radius:double=1.0 -color:String="red"
+getRadius():double +getColor():String +getArea():double

الكائن (Object): وهو عبارة عن حالة من حالات الفئة المشتق منها هذا الكائن و تتشابه في بعض الخصائص مع العديد من الكائنات المشتقة من نفس الفئة بحيث يكون لكل كائن اسم مميز له عن بقية الكائنات وخصائص خاصة به يمكن أن تتشابه مع بقية الكائنات المشتقة من نفس الفئة وافعال تمثل سلوك الكائن والشكل التالي يمثل كائن مشتق من الفئة "Circle":

```
c1:Circle
-radius=2.0
-color="blue"
+getRadius()
+getColor()
+getArea()
```

الخصائص Properties: هي مجموعة من الصفات التي تغير من مظهر الكائنات وبالرغم من أن هنالك العديد من الخصائص المشتركة بين الكائنات المختلفة إلا أنه لكل كائن من الكائنات مجموعة من الخصائص(الصفات) المميزة له عن الكائنات الاخرى ومن امثلة الخصائص (اللون، العنوان، المكان، الارتفاع، العرض ..الخ)، وفي عالم البرمجة يتم تمثيل الخصائص باستخدام متغيرات، ويمكن ضبط وتغير خصائص أي كائن المتغير الذي يمثل الخاصية.

الدوال (الطرق) Methods: وهي مجموعة الافعال التي يمكن أن تقوم بها الكائنات وهي عبارة عن دوال جاهزة تقوم بوظائف محددة مبنية داخل الفئة المشتق منها الكائن لأداء وظيفة معينة تتعلق بسلوك الكائن ويتم تنفيذ هذه الدوال باستدعائها في البرنامج من خلال كتابة اسم الكائن ثم نقطة ثم كتابة اسم الدالة (Method) وتختلف عن الخصائص في كونها لا تأخذ قيما أثناء كتابها وقد سبق وأن تم شرحها في موضوع الدوال.

الإعلان عن الفصيلة (الفئة) Class Declaration:

يتم الإعلان عن الفصيلة من خلال استخدام الكلمة المحجوزة class ثم يتبعها اسم الفصيلة ثم قوسي الفئة ليتم كتابة الخصائص والطرق بداخلها مع ملاحظة أنه يجب أن يبدأ اسم الفصيلة بحرف كبير.

```
class  Class_name {
     هنا يتم كتابة الخصائص والدوال
}
```

مثال: الإعلان عن فصيلة تحت اسم "Student" تحتوي على مجموعة من الخصائص:

```
class Student{
    String name;
    int ID;
    String Dep;
    int age;
    String address;
    double avg;
    char group;
}
```

اشتقاق كائنات من الفئة:

نلاحظ أن كافة المتغيرات (الخصائص) المعرفة في الفئة "Student" لا يمكن إعطائها قيما داخل الفئة نفسها لأن هذه المتغيرات لا تمثل طالب بحد ذاته وانما تمثل قالباً عاماً لكافة الطلاب في النظام ومن الطبيعي فإنه لا يمكن إن تتساوى قيم هذه الخصائص في كافة الطلاب ولكي نستطيع أن نعطي هذه الخصائص قيما خاصة بطالب معين يجب اولاً أن نشتق كائن (Object) أو مجموعة كائنات من هذه الفئة يمثل كل منها كائن يحمل نفس الخصائص (بقيم مختلفة) ثم نقوم بإعطاء هذه الخصائص قيماً كل حسب اسم الكائن ويمكننا برمجياً أن نشتق كائن من الفئة "Student" من خلال الكلمة السطر التالي:

```
Class_name    object_name = new class_name();
```

حيث يكتب هذا السطر في الدالة الرئيسية للبرنامج ثم يتم إعطاء القيم لكافة الخصائص من خلال كتابة اسم الكائن ثم النقطة ثم الخاصية ثم علامة التخصيص "=" ثم قيمة الخاصية على النحو التالي:

```
Object_name.field_name = value;
```

مثال: إنشاء كائن من فئة الطلاب "Student" المعرفة مسبقاً في الدالة الرئيسية لفئة أخرى تسمى "Test"

```
class Student{
    String name;
    int ID;
    String Dep;
    double avg;
    char group;
}
public class Test{
     public static void main(String args[]){
     Student st1 = new Student();
     st1.name="سالم الدروقي";
     st1.ID=52362;
     st1.Dep="الحاسوب";
     st1.avg=80;
      st1.group='c';
     }}
```

يتكون البرنامج السابقة من فئتين الأولي تحت اسم "Student" والثانية تحت اسم "Test" وهي الفئة الرئيسية في البرنامج.

ملاحظة: الفئة الرئيسية هي الفئة التي تحتوي على الدالة الرئيسية.

البرنامج السابق لن يعطي أية نتائج للمستخدم نظراً لعدم احتوائه على جملة الطباعة ولكي نستطيع طباعة البيانات السابقة فإننا سنقوم بتعريف دالة تحت اسم "print" وسنقوم بإرسال كافة البيانات

السابقة لها كبارامترات لتقوم باستقبالها وتحويلها إلى جمل الطباعة التي سيتم كتابتها بداخلها ليصبح البرنامج كاملا كما يلي:

```
public class Test
{
public static void main(String args[])
{
Student st1 = new Student(); // اشتقاق كان من الفئة Student
st1.name="سالم الدروقي";
st1.ID=52362;
st1.Dep="الحاسوب";
st1.avg=80;
st1.group='c';
st1.print(st1.name,st1.ID,st1.Dep,st1.avg,st1.group);// استدعاء دالة الطباعة
}
}
class Student    // انشاء الفئة Student
{
    String name;
    int ID;
    String Dep;
    double avg;
    char group;
void print(String p1,int p2,String p3,double p4,char p5)
  {
   System.out.println("Student name is :" + p1 );
   System.out.println("Student  id is :" + p2 );
   System.out.println("Student department is :" + p3 );
   System.out.println("Student average is :" + p4 );
   System.out.println("Student group is :" + p5 );
   }
}
```

شرح البرنامج السابق:

في السطر 14 تم إنشاء فئة تحت اسم "Student" تحتوي على مجموعة من الخصائص والتي تبدأ من السطر 16 إلى 20 ودالة واحدة تحت اسم "print" تبدأ من السطر 21 إلى 28.

في السطر رقم 5 تم اشتقاق كائن تحت اسم st1 من الفئة Student.

الاسطر من 6 إلى 10 تم فيها تخصيص قيم لكافة خصائص الكائن.

في السطر رقم 11 تم استدعاء الدالة الخاصة بالطباعة وإرسال كافة المعاملات المطلوب طباعة قيمها.

ناتج تنفيذ البرنامج :

```
Student name is :سالم الدروقي
Student  id is :52362
Student department is :الحاسوب
Student average is :80.0
Student group is :c
```

إدخال قيم الخصائص من قبل المستخدم:

نلاحظ في البرنامج السابق أنه لا يمكن أن يعطي نتائج اخرى غير النتائج التي عرضت سابقاً وذلك لأنه قد تم تخصيص قيم الخصائص في البرنامج نفسه ولإعطاء المستخدم امكانية إدخال القيم يجب استخدام جملة الإدخال بدلاً من كتابة القيم مباشرة في البرنامج (راجع جمل الادخال في لغة الجافا).

ليصبح جزء البرنامج المخصص لإدخال وتخزين القيم على النحو التالي:

```
st1.name=input.next();
st1.ID=input.nextInt();
st1.Dep=input.next();
st1.avg=input.nextDouble();
st1.group=input.next().charAt(0);
```

دلة البناء (المشيد) Constructor:

هو عبارة عن شفرة برمجية لها بداية ولها نهاية تتشابه في تركيبها مع الدالة و يتم تميزها عن بقية الدوال من خلال اسمها الذي يكون مطابقاً لاسم الفئة الموجودة فيها، كما تختلف دالة البناء عن بقية الدوال في طريقة تنفيذها حيث يتم تنفيد الشفرة البرمجية الموجودة داخلها في كل مرة يتم فيها إنشاء كائن جديد من الفئة الموجودة فيها دالة البناء، والذي يتم باستخدام الكلمة المفتاحية "new" والمستخدمة عند إنشاء كائن جديد من الفئة، كما أن دالة البناء لا يمكن أن ترجع قيمة كما هو متوفر في بعض الدوال الاخرى، و تستخدم دوال البناء في تهيئة المتغيرات المطلوب إعادة قيمتها او إعطائها قيما ابتدائية في كل مره يتم فيها إنشاء كائن جديد من الفئة، ويوجد هنالك نوعان من دوال البناء:

دوال بناء لا تحتوي على بارامترات Non Parameterized Constructor:

وهي عبارة عن دوال بناء لا تحتوي على معاملات وبالتالي فإنها لا تحتاج إلى إرسال قيم او متغيرات عند تنفيذها ويوجد منها دالة واحدة فقط في كل فئة وتسمى المشيد الافتراضي او دالة البناء الافتراضية (Default Constructor).

مثال:

```
public class Main_Class     // الفئة الرئيسية
{
    public static void main(String[] args)// الدالة الرئيسية في البرنامج
    {
     Student Std= new Student(); // Student اشتقاق كائن من الفئة
     Std.print();   // استدعاء دالة الطباعة
    }
}
class Student   // الفئة
{
  public String name="Ahmed"; //name تعريف المتغير

  public Student()
  {
    name="Salem"; //name تهيئة المتغير
  }

     public void print()// دالة الطباعة
     {
         System.out.println("welcome Mr "+name);
     }
 }
```


في المثال السابق تم إنشاء مشروع يحتوي على عدد فئتين الأولى هي الفئة الرئيسة والمسماة (Main_class) وهي الفئة التي تحتوي على الدالة الرئيسية "main" والثانية هي الفئة "Student" وتحتوي على جملة لتعريف متغير من نوع سلسلة حرفية (String) تحت اسم "name" وإعطائة قيمة ابتدائية "Ahmed" كما تحتوي هذه الفئة على دالة البناء (Student) والتي تم فيها تهيئة المتغير"name " وإعطائه قيمة "Salem" بالإضافة إلى ذلك فإن الفئة " Student" تحتوى على دالة الطباعة (Print) لطباعة قيمة المتغير (name).

عند تنفيذ البرنامج فإن المترجم يبدأ مباشرة في تنفيذ الشفرة البرمجية الموجودة داخل الدالة الرئيسية في البرنامج بحيث يتم اشتقاق كائن من الفئة Student تحت اسم " Std " كما يتم في نفس السطر استدعاء المشيد "Student" من خلال وجود الكلمة المفتاحية "new" في نفس السطر ثم يتم بعدها استدعاء الدالة Print الموجودة في الفئة Student من خلال الكائن "Std" ليتم تنفيذ الشفرة البرمجية الموجودة بداخلها و التي ينتج عنها طباعة الجملة التالية(welcome Mr Salem) على شاشة الطباعة.

من خلال ناتج تنفيذ البرنامج السابق نستطيع أن نلاحظ فكرة عمل المشيد، حيث أنه وبالرغم من أن قيمة المتغير name هي "Ahmed" إلا أنه عند اشتقاق كائن من الفئة تم تنفيذ الشفرة البرمجية الموجودة بداخل المشيد المسمى "Student" والتي قامت بتغيير قيمة المتغير name من Ahmed إلى Salem.

دوال بناء تحتوي على بارمترات Parameterized Constructor:

وهي عبارة عن دوال بناء تحتوي على معاملات وبالتالي فإنها تحتاج إلى إرسال قيم او متغيرات عند تنفيذها ويمكن للمبرمج تعريف اكثر من دالة بناء (مشيد) في نفس الفئة مع الأخذ في الاعتبار بأن هذه المشيدات جميعها يجب أن تحمل نفس اسم الفئة على أن تختلف في عدد المعاملات (البارامترات) أو انواع المعاملات أو ترتيب أنواع هذه المعاملات لكي يتم التميز بينها أثناء استدعائها وهذا ما يعرف بـ التحميل الزائد للمشيد، وقد سبق وأن تم توضيح هذا المفهوم في موضوع التحميل الزائد للدوال.

استدعاء المشيدات Constructor Invocation:

بالرغم من وجود اوجه تشابه بين المشيدات والدوال إلا أنها تختلف في طريقة استدعائها، فبينما تستدعى الدوال من خلال كتابة اسمها في البرنامج فإن المشيدات تستدعى في كل مره يتم فيها إنشاء كائن جديد من الفئة الموجود بها المشيد وذلك من خلال الكلمة المفتاحية new، كما أن استدعاء أنواع المشيدات يختلف في حالة كان المشيد المطلوب استدعائه مشيد افتراضي ام لا وفيما يلي نقدم الشكل العام لكلا النوعين:

الشكل العام لاستدعاء المشيد الافتراضي Default Constructor Invocation:

لاستدعاء المشيد الذي لا يحتوي على معاملات (المشيد الافتراضي) يتم كتابة اسم الفئة ثم اسم الكائن ثم علامة "=" ثم الكلمة المحجوزة "new" ثم اسم الفئة على النحو التالي:

```
class_name  object_name = new  class_name
```

ملاحظة : من خلال السطر السابق نلاحظ ان عملية استدعاء المشيد الافتراضي تتم في كل مرة يتم فيها انشاء كائن جديد من الفئة الموجود بها المشيد.

الشكل العام لاستدعاء مشيد غير افتراضي Parameterized Constructor Invocation:

لاستدعاء أي مشيد أخر غير المشيد الافتراضي (المشيدات البارامترية) والذي سيكون بالضرورة محتوياً على معامل (بارامتر) واحد على اقل تقدير، يتم كتابة اسم الفئة ثم اسم الكائن ثم علامة "=" ثم الكلمة المحجوزة "new" ثم اسم الفئة متبوعاً بمجموعة المعاملات (متغيرات او ثوابت) بحيث توضع بين قوسين تفصل بينها فاصلة على النحو التالي:

```
class_name  object_name = new  class_name(param1,param2,....)
```

ملاحظة: لا يمكن للمبرمج تعريف مشيد او أكثر من المشيدات الغير الافتراضية (المشيدات البارامترية) دون تعريف المشيد الافتراضي.

مثال : المثال التالي يوضح طريقة تعريف واستدعاء المشيد الافتراضي والمشيدات البارامترية الاخرى:

```
class Example{

  Example()  // تعريف المشيد الافتراضي
   {
    System.out.println("Default constructor");
   }

  Example(int i, int j)  // تعريف مشيد يحتوي على معاملين
   {
     System.out.print("parameterized constructor");
     System.out.println(" with Two parameters");
   }

  Example(int i, int j, int k)  // تعريف مشيد يحتوي على ثلاثة معاملات
   {
     System.out.print("parameterized constructor");
     System.out.println(" with Three parameters");
   }

  public static void main(String args[]) // الدالة الرئيسية
   {
     // استدعاء المشيد الافتراضي
    Example obj = new Example();

    //استدعاء المشيد الذي يحتوي على معاملين//
    Example obj2 = new Example(12, 12);

    // استدعاء المشيد الذي يحتوي على ثلاث معاملات
    Example obj3 = new Example(1, 2, 13);
   }
}
```

استدعاء مشيد من داخل مشيد آخر Calling one constructor from another:

تستخدم الكلمة المفتاحية "this" لاستدعاء مشيد من داخل مشيد آخر في نفس الفئة مع الأخذ في الاعتبار ضرورة كتابة جملة الاستدعاء باستخدام الكلمة المفتاحية "this" في بداية المشيد (اول سطر في المشيد) على النحو التالي :

```
this();        // في حالة استدعاء المشيد الافتراضي من اي مشيد اخر
this(param1,param2,....);        // في حالة استدعاء المشيدات البارامترية
```

مثال:

```
public class Book {

 public Book ()
   {
         System.out.println("Default constructor");
   }

  public Book (String str)
   {
         this();// استدعاء المشيد الافتراضي
        System.out.println("Parametrized constructor with single param");
   }

public Book (String str, int num)
   {
      this("Hello");  // استدعاء المشيد الاول
      System.out.println("Parametrized constructor with double args");
   }

public Book (int num1, int num2, int num3)
  {
     this("Hello", 2); // استدعاء المشيد الثاني
     System.out.println("Parametrized constructor with three args");
  }
```

```
public static void main(String args[]){
        // إنشاء كائن باستخدام المشيد الثالث
        Book obj = new Book (5,5,15);
   }
}
```

مخرجات البرنامج

```
Default constructor
Parametrized constructor with single param
Parametrized constructor with double args
Parametrized constructor with three args
```

الوراثة Inheritance:

يعتبر مصطلح الوراثة من أهم المفاهيم الأساسية في البرمجة الموجهة بالهدف (OOP) وهي تشابه إلى حد كبير مصطلح الوراثة الموجود في حياة الكائنات الحية، حيث يمكن لكائن حي (ابن) أن يرث الخصائص والأفعال من كائن حي أخر (اب)، ايضاً فإن البرمجة الموجهة بالهدف تمكننا من تعريف فئة ابن (subclass) قادرة على وراثة بعض الخصائص والطرق من فئة اب (superclass) من خلال استخدام الكلمة المفتاحية Extend أثناء تعريف الفئة الابن، كما يجدر بنا الإشارة إلى أنه يمكن أن نقوم باشتقاق أكثر من فئة ابن من فئة اب واحدة والعكس غير صحيح، إلا أنه لا يمكن للفئة الابن وراثة الطرق والخصائص الموجودة في الفئة الاب إلا اذا كان مستوى الوصول لها عام (public) او محمي (protected).

الشكل العام للإعلان عن فئة اب وفئة ابن مشتقة منها:

```
class Super_Class_Name       // الفئة الاب
{
   //Members and methods declarations.
}

class Sub_Class_name extends Super_Class_name //الفئة الابن
{
  //Members and methods declarations.
}
```

في المثال السابق تم الاعلان عن فئة اب من خلال الكلمة المحجوزة class تحت اسم Super_Class_Name كما تم الاعلان عن فئة ابن باستخدام الكلمة المحجوزة class تحت اسم Sub_Class_name متبوعة بالكلمة المحجوزة extends ثم كتابة اسم الفئة الاب المشتقة منها.

مثال :

```
class Vehicle {

 String color;
   int speed;
   int size;

   void attributes()
   {
      System.out.println("Color : " + color);
      System.out.println("Speed : " + speed);
      System.out.println("Size : " + size);
   }
}
class Car extends Vehicle {
   int CC;
   int gears;
   void attributescar() {
    System.out.println("Color of Car : " + color);
    System.out.println("Speed of Car : " + speed);
    System.out.println("CC of Car : " + CC);
    System.out.println("No of gears of Car : " + gears);
   }
}
public class Test {
    public static void main(String[] args) {

  Car Nissan = new Car(); // اشتقاق كائن من الفئة الابن
      Nissan.color = "Blue";
      Nissan.speed = 200 ;
      Nissan.CC = 1000;
      Nissan.gears = 5;
      Nissan.attributescar();

  Vehicle Hyundai = new Vehicle ();// اشتقاق كائن من الفئة الاب
      Hyundai.color="red";
      Hyundai.speed=220;
      Hyundai.attributes();
    }
}
```

الفئة الاب

الفئة الابن

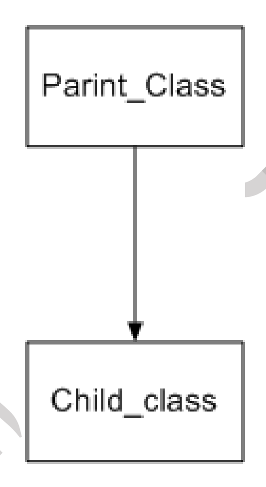
الفئة الرئيسية

أنواع الوراثة في لغة الجافا Types of inheritance:

توجد في لغة الجافا عدة أنواع من الوراثة تتمثل في الوراثة الاحادية، الوراثة متعددة المستويات، الوراثة الهرمية فيما يلي نقدم شرحاً بالأمثلة لهذه الانواع:

الوراثة الاحادية Single Inheritance:

تعتبر الوراثة الاحادية من ابسط أنواع الوراثة واكثرها سهولة واستخداماً حيث يتم فيها اشتقاق فئة ابن واحدة من فئة أب والشكل التالي يوضح فكرة الوراثة الاحادية:



كما يوضح المثال التالي طريقة تطبيق الوراثة الاحادية باستخدام لغة الجافا:

```
Class A    // فئة اب
{
   public void method_A()
   {
     System.out.println("Parint class method");
   }
}
Class B extends A  // فئة ابن
{
   public void method_B()
   {
     System.out.println("Child class method");
   }
```

```
public static void main(String args[]) // الدالة الرئيسية
   {
     B obj = new B();  //   اشتقاق كائن من الفئة الابن
     obj.method_A(); //استدعاء دالة الفئة الاب
     obj.method_B(); //استدعاء دالة الفئة الابن
  }
}
```

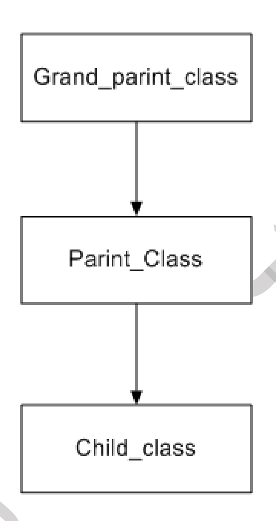
ملاحظة : من خلال المثال السابق نلاحظ أنه وبالرغم من أن الكائن "obj" تم اشتقاقه من الفئة الابن (B) إلا أننا تمكنا من خلاله من استدعاء الدالة "method_A " الموجودة في الفئة الاب (A).

في المثال السابق تم تعريف فئة أب تحت اسم "A " تحتوي على دالة واحدة تحت اسم method_A

كما تم تعريف فئة ابن تحت اسم "B" تحتوي على دالة واحدة تحت اسم method_B، بالإضافة إلى الدالة الرئيسة في البرنامج التي تم فيها اشتقاق كائن تحت اسم "obj" من الفئة "B" والذي تم من خلاله استدعاء الدالة method_A الموجودة في الفئة الاب وكذلك الدالة method_B الموجودة في الفئة الابن، وبهذا فإنه يمكن القول بأن الفئة الابن (B) قامت بوراثة الفئة الاب (A) من خلال امكانية استدعائها للدالة الموجودة في الفئة الاب "A " من خلال الكائن "obj" المشتق من الفئة الابن "B"، ومن هنا يتضح مفهوم الوراثة في البرمجة الموجهة من خلال قدرة كائن مشتق من الفئة الابن من استدعاء دالة موجودة في الفئة الاب مع اشتراط أن يكون مستوى الوصول للدالة من نوع عام (public) او من نوع محمي (Protected).

الوراثة متعددة المستويات Multilevel inheritance:

يوجد هنالك تشابه إلى حد كبير بين الوراثة متعددة المستويات والوراثة الاحادية مع وجود اختلاف في امكانية اشتقاق فئة ابن اخرى من الفئة الابن الأولى لتصبح الفئة الابن الأولى فئة أب للفئة الجديدة، ليصبح التسلسل المنطقي لعملية الوراثة إن صح التعبير (فئة جد، فئة أب، وفئة ابن) والشكل التالي يوضح طريقة تمثيل الوراثة متعددة المستويات:



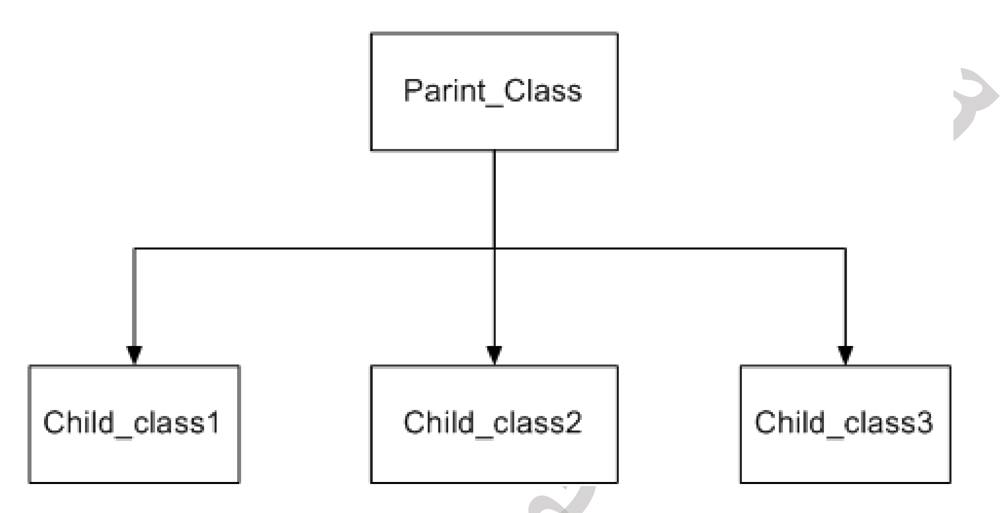
والمثال التالي يوضح طريقة تطبيق الوراثة متعددة المستويات باستخدام لغة الجافا:

```
Class X{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
       public void methodY()
       {
       System.out.println("class Y method");
       }
}
Class Z extends Y{
   public void methodZ(){
     System.out.println("class Z method");
   }

public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //استدعاء دالة الفئة الجد
     obj.methodY(); //استدعاء دالة الفئة الاب
     obj.methodZ(); //استدعاء دالة الفئة الابن
  }
}
```

الوراثة الهرمية Hierarchical Inheritance :

تتمثل الوراثة الهرمية في امكانية اشتقاق أكثر من فئة ابن من فئة أب واحدة والشكل التالي يوضح فكرة الوراثة الهرمية:



والمثال التالي يوضح طريقة تطبيق الوراثة الهرمية باستخدام لغة الجافا:

```
Class A{
  public void methodA()
  {
     System.out.println("method of Class A");
  }
}

Class B extends A{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
}

Class C extends A{
 public void methodC()
 {
 System.out.println("method of Class C");
 }
}


Class D extends A{
  public void methodD()
  {
     System.out.println("method of Class D");
  }
}

Class MyClass{
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

ملاحظة : توجد في البرمجة الموجهة انواع اخرى من الوراثة مثل الوارثة المتعددة والوراثة الهجينة إلا أنه لا يمكن تطبيقها في لغة الجافا إلا من خلال آلية الوسط البيني (interface) التي سيتم شرحها لاحقا.

الكلمة المفتاحية super:

تستخدم الكلمة المفتاحية "super" للوصول للبيانات (المتغيرات) والدوال والمشيدات الموجودة في الفئة الاب من داخل الفئة الابن على النحو التالي:

استدعاء البيانات في الفئة الاب:

يمكن استخدام الكلمة المفتاحية "super" للوصول إلى البيانات والمتغيرات (الخصائص) الموجودة في الفئة الاب من داخل الفئة الابن خصوصا عند وجود تشابه بين أسماء المتغيرات وانواعها في كل من الفئة الاب والفئة الابن وكمثال على ذلك:

```
class Human // الفئة الاب
{
    //متغيرات الفئة الاب
    String name="Ahmed";
    int id=10;
}

public class Student extends Human //  الفئة الابن
{
    String name="Salem";
    int id=520;

    void print()
    {
          // طباعة المتغيرات  الموجود في الفئة الاب
        System.out.println(super.name);
        System.out.println(super.id);
    }
   public static void main(String[] args)
    {
        Student obj=new Student(); // اشتقاق كائن من الفئة الابن
        obj.print();    // استدعاء الدالة الموجودة في الفئة الابن
        }
 }
```

ناتج تنفيذ البرنامج :

```
Ahmed
10
```

استدعاء الدوال في الفئة الاب من مشيد الفئة الابن:

تستخدم الكلمة المفتاحية super في استدعاء الدوال المكررة الدالة الموجودة في الفئة الاب من داخل مشيد الفئة الابن او من داخل أي دالة اخرى حيث تكتب على النحو التالي:

```
super.method_name();
```

على أن تكتب في اول سطر من مشيد الفئة الابن.

```
class Device
{
       Device ()
       {
           System.out.println("defult Parent Constructor");
       }
       Device (int x)
       {
           System.out.println("Second Parent Constructor");
       }
      void print()
      {
       System.out.println("Method in Parent class");
       }
 }

public class Computer extends Device
{

       Computer()
        {

           super.print(); // استدعاء الدالة print في الفئة الاب
        }
    public static void main(String[] args)
    {
     Computer obj=new Computer();
    }
 }
```

مثال : المثال التالي يوضح استدعاء دالة (غير المشيد) في الفئة الاب من خلال الكلمة المفتاحية super من خلال دالة في الفئة الابن.

```
class Animal{

   public void move()
   {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal{

   public void move()
   {
      super.move();   // invokes the super class method
      System.out.println("Dogs can walk and run");
   }
}

public class TestDog{

   public static void main(String args[]){

      Animal b = new Dog(); // Animal reference but Dog object
      b.move(); //Runs the method in Dog class

   }
}
```

استدعاء المشيد في الفئة الاب:

تستخدم الكلمة المفتاحية super في استدعاء مشيد الفئة الاب من داخل مشيد الفئة الابن حيث تكتب على النحو التالي:

```
super();
```

على أن تكتب في أول سطر من مشيد الفئة الابن و إلا ستظهر رسالة الخطأ التالية :

“Constructor call must be the first statement in a constructor”

مثال:

```
class Device
{
 Device()
 {
     System.out.println("Parent Constructor");
 }

}
public class Computer extends Device
{
  Computer()
  {
      super(); //  استدعاء مشيد الفئة الاب
      System.out.println("child Constructor");
  }
    public static void main(String[] args)
    {
      Computer obj=new Computer();// استدعاء مشيد الفئة الابن
    }
 }
```

كما يمكننا من خلال الكلمة المفتاحية super استدعاء المشيدات البارامترية (المشيدات التي تحتوي على معاملات) من خلال ارفاق قيمة المعامل بين القوسين امام الكلمة المفتاحية super على النحو التالي:

```
super(value);
```

مثال:

```
class Computer
{
       Computer ()
       {
           System.out.println("defult Parent Constructor");
       }
       Computer (int x)
       {
           System.out.println("Second Parent Constructor "+x);
       }

}
public class Laptop extends Computer
{

  Laptop()
  {
      super(2);
  }

 public static void main(String[] args)
    {
       Laptop obj=new Laptop();
    }
 }
```

مخرجات البرنامج

```
Second Parent Constructor 2
```

تعدد الاوجه (الاشكال) Polymorphism:

يعتبر مصطلح تعددية الاوجه او تعددية الاشكال من أهم المصطلحات المستخدمة في البرمجة الموجهة بالهدف (OOP) ويعني امكانية أن تقوم الدالة بأفعال مختلفة اعتماداً على الكائن الذي يتم استدعائها من خلاله ويمكن تمثيل ذلك من خلال نوعين من تعددية الاوجه وهما التحميل الزائد للدوال (Method Overloading) و تجاهل الدالة (Method Overriding).

أنواع تعدد الاشكال Types of polymorphism:

يوجد هنالك نوعان من تعددية الاشكال في لغة الجافا وهما:

تعددية الأوجه الثابتة:.(Compile time polymorphism (static polymorphism:

يتمثل هذا النوع من تعددية الاوجه في التحميل الزائد للدالة Method Overloading والذي يتمثل في إمكانية تعريف أكثر من دالة بنفس الاسم في نفس الفئة على أن تختلف هذه الدوال في ما يعرف بتوقيع الدالة والذي يتمثل في (عدد معاملات الدالة، أنواع معاملات الدالة، ترتيب أنواع معاملات الدالة).

تعددية الاوجه المتغيرة (Runtime polymorhism(Dynamic polymorphism:

يتمثل هذا النوع من تعددية الاوجه في تجاهل الدالة Method Overriding، ويختلف مصطلح Method Overriding عن مصطلح Method Overloading الذي تم شرحه مسبقاً والذي يعني امكانية تعريف فئة تحتوي على اكثر من دالة بنفس الاسم يتم التميز بينها من خلال ما يعرف بتوقيع الدالة بينما يعني مصطلح Method Overriding امكانية تعريف دالة في فئة أب

وأخرى في فئة ابن مشتق منه بنفس الاسم بحيث يتم التمييز بينها من خلال مرجعية الكائن الذي يستدعي هذه الدالة.

مثال:

```
class Human{
   public void eat()
   {
      System.out.println("Human is eating");
   }
}
class Boy extends Human{
   public void eat()
   {
      System.out.println("Boy is eating");
   }
   public static void main( String args[])
   {
      Boy obj = new Boy();
      obj.eat();
   }
}
```

إرسال الطرق ديناميكيا Dynamic method dispatch:

هي تقنية نستطيع من خلالها إنشاء كائن من الفئة الابن تكون مرجعتيه إلى الفئة الاب، بحيث يستطيع هذا الكائن استدعاء كافة الدوال المشتركة بين الفئة الاب والفئة الابن كما يستطيع استدعاء كافة الدوال الموجودة في الفئة الاب سواء كانت مشتركة ام لا ، إلا أنه لا يستطيع استدعاء الدوال الجديدة الموجودة في الفئة الابن (الدوال الغير مشتركة والموجودة في الفئة الابن).

```
class ABC{
   public void disp()
   {
      System.out.println("disp() method of parent class");
   }
   public void abc()
   {
      System.out.println("abc() method of parent class");
   }
}
class Test extends ABC{
   public void disp(){
      System.out.println("disp() method of Child class");
   }

   public void xyz(){
      System.out.println("xyz() method of Child class");
   }
   public static void main( String args[]) {
      //Parent class reference to child class object
      ABC obj = new Test();
      obj.disp();//
      obj.abc();
   }
}
```

الكبسلة Encapsulation:

بالإضافة إلى مصطلح الوراثة وتعددية الاوجه يعتبر مصطلح الكبسلة (اخفاء البيانات) من المفاهيم الأساسية للبرمجة الموجهة بالهدف، ويقصد بها عملية إخفاء البيانات والمعلومات الموجودة في كل فئة عن الفئات الاخرى بحيث لا يمكن لأي فئة الوصول إلى البيانات الموجودة في الفئات الاخرى إلا من خلال إرسال رسائل إلى دوال عامة تقوم بالوصول إلى هذه البيانات وإرجاع النتيجة إلى هذه الفئة، هذا من ناحية ومن ناحية اخرى يمكن تعريف اخفاء البيانات بأنه عملية اخفاء تفاصيل التنفيذ عن المستخدم.

ومن منطلق برمجي فإنه يمكن تعريف عملية اخفاء البيانات بأنه عملية تعريف بيانات (متغيرات) (خصائص) داخل فئة معينة بمستوى وصول خاص (private) بحيث لا يمكن الوصول إليها إلا من داخل الفئة المعرفة فيها ولتوضيح الفكرة نقوم بتقديم المثال التالي:

```
class   Student{
  private name ;
  private age;
  private address;
}
```

في المثال السابق تم تعريف فئة تحمل اسم Student تحتوي على ثلاث خصائص بمستوى وصول خاص (لا يمكن الوصول لها إلا من خلال الفئة نفسها)، وبالتالي فإنه في حالة اشتقاق كائن من الفئة Student في فئة اخرى (الفئة الرئيسية) فإنه لا يمكن الوصول إلى هذه البيانات من خلال الكائن المشتق من الفئة Student.

```
class Student{
     private String name ;
     private int age;
     private String address;

}
public class Test {
    public static void main(String[] args) {
    Student std=new Student();
    std.name="Salem";
    std.age=33;
    std.address="Alkhoms";
    }
 }
```

في المثال السابق تم اشتقاق كائن (std) من الفئة Student داخل الدالة الرئيسية الموجودة في الفئة Test كما تم تخصيص قيم للخصائص (name,age,address) الموجودة في الفئة Student من خلال كتابة اسم الكائن std متبوعا باسم الخاصية ، إلا أنه عند تنفيذ البرنامج فانة سيظهر لنا رسالة الخطأ التالية :

Uncompilable source code - name has private access in Test.Student
at Test.Test.main(Test.java:15)

حيث تفيد الرسالة السابقة بأن مستوي الوصول إلى الخاصية name في الفئة Student من النوع الخاص (private) ولا يمكن الوصول إليها من الدالة الرئيسية للفئة Test، ولحل هذه المشكلة يتم استخدام دوال ذات مستوى وصول عام (Public) تعرف داخل الفئة (Student) التي تحتوي على البيانات المطلوب الوصول إليها حيث يتم استدعاء هذه الدوال من قبل الكائن المشتق من الفئة لتقوم بتحديث قيم البيانات المخفية وكذلك عرض هذه القيم وهذه الدوال تعرف بدوال getter,setter.

دوال الـ Setter , Getter:

تعتبر دوال الـ Setter وGetter من اشهر الدوال المستخدمة للوصول إلى البيانات المخفية في فئة معينة بحيث يتم تعريف هذه الدوال على أساس دوال عامة (Puplic) بحيث تقوم الدالة setter بتخصيص قيم للخصائص المخفية (name , age , address) داخل الفئة (Student)، كما تقوم الدالة getter بعرض قيم للخصائص المخفية (name , age , address) داخل الفئة (Student) ليصبح المثال السابق على النحو التالي

```
class Student{
      private String name ;
      private int age;
      private String address;

      // دالة لتحديث الاسم
      public void setname(String newValue ){
      name = newValue;
       }

      // دالة لتحديث العمر
      public void setAge(int newValue ){
      age = newValue;
      }

      // دالة لتحديث العنوان
      public void setAddress(String newValue ){
      address = newValue;
      }


      // دالة لارجاع الاسم
      public String getname(){
      return name;
      }
```

```
    // دالة لارجاع العمر
    public int getAge(){
    return age;
    }
    // دالة لارجاع العنوان
    public String getAddress(){
    return address;
    }
}
public class Test {
    public static void main(String[] args) {
    Student std=new Student();  // اشتقاق كائن من الفئة
    std.setname("Salem");      // استدعاء دالة تحديد الاسم
    std.setAge(33);           // استدعاء دالة تحديد العمر
    std.setAddress("alkoms");// استدعاء دالة تحديد العنوان

 // استدعاء دالة ارجاع الاسم وطباعة الاسم
System.out.println("Student Name is : " + std.getname());

 // استدعاء دالة ارجاع العمر وطباعة العمر
System.out.println("Student Age is: " + std.getAge());

 // استدعاء دالة ارجاع العنوان وطباعة العنوان
System.out.println("Student Address is : " + std.getAddress());
    }
}
```

مخرجات البرنامج

```
Student Name is : Salem
Student Age is: 33
Student Address is : alkoms
```

من خلال المثال السابق يتضح لنا أن البيانات السابقة المعرفة داخل الفئة Student عبارة عن بيانات مخفية لا يمكن الوصول إليها بشكل مباشر نظرا لاستخدام تقنية اخفاء البيانات ولذلك استخدمت دوال التحديد (setter) ودوال الحصول (getter) للوصول إلى هذه البيانات المخفية.

إنشاء الفئات المتداخلة:

هي امكانية تعريف فئة داخل فئة اخرى بغية الوصول إلى بعض الخصائص بين هذه الفئات وتوجد هنالك نوعان من الفئات المتداخلة وهي الفات المتداخلة الثابتة Static nested class والفئات المتداخلة الغير ثابتة Non-static nested class.

الفئات المتداخلة غير الثابتة Non-static nested class:

تصنف الفئات المتداخلة غير الثابتة إلى ثلاثة انواع اعتماداً على كيفية ومكان تعريف الفئة الداخلية وهذه الانواع هي:

الفئة الداخلية inner classes:

ويتم فيها تعريف الفئة الداخلية مباشرةً داخل اقواس الفئة الخارجية.

الفئة الداخلية المحلية في الدالة Method -Local inner classes:

وكما يدل اسمها فإنه يتم فيها تعريف الفئة الداخلية داخل نطاق دالة موجودة في الفئة الخارجية وتعتبر هذه الفئة فئة محلية خاصة بالدالة لا تتعدى حدودها حدود الدالة.

الفئة الداخلية المجهولة Anonymous Inner Classes:

هي إحدى طرق تطبيق تقنية الفئات المتداخلة وتتميز بعدم وجود اسم للفئة الداخلية ولهذا سميت بالفئة المجهولة، ولا يمكن اشتقاقها إلا مرة واحدة فقط في نفس وقت تعريفها (تعريف الفئة) وتنتهي هذه الفئة بفاصلة منقوطة.

الفئات المتداخلة الثابتة Static Nested Classes :

وهي عبارة عن فئة داخلية يتم تعريفها على أساس أنها عضو ثابت في الفئة الخارجية بحث يمكن الوصول إليها دون اشتقاق كائن من الفئة الخارجية.

وسيتم في هذا الكتاب دراسة النوع الاول من الفئات المتداخلة غير الثابتة بشيء من التفصيل.

الفئة الداخلية Inner classes

الفئة الداخلية (Inner class) هي فئة يتم تعريفها داخل جسم فئة اخرى مباشرة والتي تسمى الفئة الخارجية (Outer class) بحيث تتكون بينهما علاقة خاصة تسمح للفئة الداخلة من الوصول إلى كافة اعضاء الفئة الخارجية (المتغيرات و الدوال) ، حتى وإن كان مستوى الوصول لها من نوع خاص (private) ويوجد عدة انواع من الفئات الداخلية تصنف حسب طريقة تعريفها في البرنامج والمثال التالي يوضح فكرة الفئات المتداخلة.

```
public class Book{   // بداية الفئة الخارجية
         class pen  { //بداية الفئة الداخلية
                    // هنا يتم كتابة شفرة الفئة الداخلية
                    }// نهاية الفئة الداخلية
 }// نهاية الفئة الخارجية
```

في المثال السابق تم تعريف فئة خارجة تحت اسم Book وفئة داخلية تحت اسم Pen بحيث يمكن للفئة الداخلية (Pen) من الوصول إلى المتغيرات والدوال الموجودة في الفئة الخارجية (Book) ولتطبيق ذلك نقوم بتعديل المثال السابق ليصبح على النحو التالي:

```
public class Book {  // بداية الفئة الخارجية
    private int outerIntVar=10;
    private String outerStrVar="Salem";

    class pen{ //بداية الفئة الداخلية
          public void print()
          {
                System.out.print("Outer class int variable" +outerIntVar);
                System.out.print("Outer class String Variable"+outerStrVar);
          }
    }// نهاية الفئة الداخلية
    public static void main(String[] args) {

    }
   }// نهاية الفئة الخارجية
```

في المثال السابق والذي يوضح جزء من برنامج تم تعريف متغيرين مستوى الوصول لهما من نوع خاص (private) داخل الفئة الخارجية مع تخصيص قيمة ابتدائية لكل متغير اثناء التعريف، كما تم تعريف دالة داخل الفئة الداخلية تقوم بطباعة قيم المتغيرات الخاصة المعرفة في الفئة الرئيسية والتي لا يمكن الوصول إليهما بشكل مباشر من اي فئة اخرى من خلال تطبيق فكرة الفئات المتداخلة.

اشتقاق كائن من الفئة الداخلية:

تختلف طريقة اشتقاق كان من الفئة الداخلية حسب طريقة كتابة الشفرة البرمجية الخاصة باشتقاق هذا الكائن بحيث يمكن كتابة شفرة الاشتقاق داخل دالة يتم تعريفها في الفئة الخارجية كما يمكن كتابة شفرة الاشتقاق في الدالة الرئيسية للبرنامج.

اولا اشتقاق كائن من الفئة الداخلية من داخل الشفرة البرمجية لدالة في الفئة الخارجية

لكي نتمكن من اشتقاق كائن من الفئة الداخلية يجب اولاً اشتقاق كائن من الفئة الخارجية لكي نستطيع من خلاله اشتقاق كائن من الفئة الرئيسية والمثال التالي يوضح طريقة اشتقاق كائن من الفئة الداخلية.

```
public class outerClass {// بداية الفئة الخارجية
   private int outerIntVar=10;
   private String outerStrVar="Salem";

  public void InstOuter(){ // دالة لاشتقاق كائن من الفئة الداخلية
       innerClass inner=new innerClass();
       inner.print();
   }
   class innerClass{ //بداية الفئة الداخلية
   public void print(){
    System.out.println("Outer class int variable =" +outerIntVar);
     System.out.println("Outer class String Variable ="+outerStrVar);
    }

  }// نهاية الفئة الداخلية
   public static void main(String[] args) {
   outerClass BK=new outerClass();
   BK.InstOuter();
   }
  }// نهاية الفئة الخارجية
```

شرح المثال السابق

في المثال السابق تم تعريف فئة خارجية تحت اسم outerClass كما تم تعريف فئة داخلية تحت اسم innerClass بحيث :

تحتوي الفئة الخارجية على:

– تعريف لمتغيرين الاول من النوع الصحيح (int) والثاني من نوع السلسلة (String) و مستوى الوصول لكل منهما من نوع خاص (private).

- تعريف لدالة تحت اسم InstOuter بحيث تحتوي هذه الدالة على شفرة برمجية تقوم باشتقاق كائن (Object) تحت اسم inner من الفئة الداخلية innerClass، كما تحتوي على سطر لاستدعاء الدالة الموجودة في الفئة الداخلية.

– الدالة الرئيسية في البرنامج والتي تحتوي على شفرة برمجية تقوم باشتقاق كائن (Object) تحت اسم BK من الفئة الداخلية outerClass ، كما تحتوي على سطر لاستدعاء الدالة InstOuter الموجودة في الفئة الخارجية .

و تحتوي الفئة الداخلية على:

– تعريف لدالة تقوم بطباعة قيم المتغيرات الموجودة في الفئة الخارجية عند استدعائها.

تسلسل تنفيذ البرنامج :

ببساطة ودون الدخول في التفاصيل يمكن توضيح تسلسل تنفيذ البرنامج السابق من خلال الخطوات التالية :

يبدأ تنفيذ البرنامج من السطر رقم 14 الذي يتم فيه اشتقاق كائن من الفئة الخارجية ثم ينتقل إلى السطر رقم 15 الذي يتم فيه استدعاء الدالة الموجودة في السطر رقم 4 لينتقل التنفيذ إلى السطر رقم 5 والذي يتم فيه اشتقاق كائن من الفئة الداخلية ثم ينتقل التنفيذ إلى السطر رقم 6 والذي يتم فيه استدعاء الدالة المعرفة في الفئة الداخلية في السطر رقم 9 لينتقل التنفيذ إلى السطر رقم

10 ثم السطر رقم 11 واللذان يتم فيهما طباعة قيم المتغيرات المعرفة مسبقا في السطر رقم 2 والسطر رقم 3 في الفئة الخارجية ثم يتم عرض النتائج على الشاشة.

اشتقاق كائن من الفئة الداخلية داخل الدالة الرئيسية في الفئة الخارجية:

في هذه الطريقة يتم اشتقاق كائن من الفئة الداخلية مباشرةً داخل الدالة الرئيسية بحيث يتم الاستغناء عن الدالة المعرفة في الفئة الخارجية والتي كانت تستخدم لاشتقاق كائن من الفئة الداخلية ليصبح البرنامج كما يلي:

```
public class OuterClass {

    private int outerIntVar=10;
    private String outerStrVar="Salem";

    class innerClass{
    public void print(){
          System.out.println("Outer class int variable =" +outerIntVar);
         System.out.println("Outer class String Variable="+outerStrVar);
    }
    }
  public static void main(String[] args) {
         OuterClass.innerClass inner = new OuterClass().new innerClass();
  inner.print();
   }
}
```

التجريد Abstraction:

يعتبر التجريد احد المفاهيم الأساسية في تقنية البرمجة الموجهة بالهدف ويمكن تعريف التجريد بأنه عملية ازالة التفاصيل واخفائها قدر الامكان مع الابقاء على صحة المعلومة، ويمكن تطبيق تقنية التجريد في لغة الجافا من خلال الفئة المجردة (Abstract Class)، والوسط البيني(Interface)، حيث يمكننا الوسط البيني من تطبيق تقنية التجريد بنسبة 100%، بينما تمكننا الفئة المجردة من تطبيق تقنية التجريد بنسبة تتراوح بين 1 إلى 100%.

الفئات والدوال المجردة:

الفئات المجردة Abstract Classes:

الفئة المجردة هي عبارة عن فئة يتم تعريفها مسبوقة بالكلمة المفتاحية abstract لتدل على أن هذه الفئة هي فئة مجردة ، وتختلف عن بقية الفئات في عدم امكانية اشتقاق كائن من الفئة المجردة وبالتالي فإنها يجب أن تورث من فئة اخرى بحيث تكون لها فئة ابن واحدة على الاقل يتم فيها تطبيق الدوال المجردة في الفئة المجردة (الفئة الاب) ، كما يمكن أن تحتوي الفئة المجردة على دوال مجردة (Abstract Method) واخرى غير مجردة (Concrete Methods) ، كما أنه في حالة الرغبة في تعريف دالة واحدة مجردة داخل الفئة فإن هذه الفئة يجب أن يتم تعريفها على اساس فئة مجردة.

الشكل العام لتعريف الفئة المجردة:

برمجيا يمكننا انشاء الفئة المجردة بنفس الطريقة التي يتم بها انشاء الفئة المكتملة على أن تكون مسبوقة بالكلمة المفتاحية abstract وفيما يلي نقدم الشكل العام لإنشاء الفئة المجردة:

```
abstract Accessmodifier Abstract_class_name
{
  ......
}
```

مثال: يوضح هذا المثال عملية انشاء فئة مجردة تحت اسم Animal :

```
abstract  public Animal
{

}
```

الدوال المجردة Abstract Methods:

هي عبارة عن دوال يتم تعريفها باستخدام الكلمة المفتاحية abstract داخل الفئات المجردة، تحتوي فقط على راس الدالة، ولا يمكن أن تحتوي على جسم الدالة وينتهي تعريفها بالفاصلة المنقوطة، وكما أن الفئة المجردة يجب أن تكون لها فئة ابن واحدة على الاقل، فإن الدالة المجردة يجب أن تكون مكررة في الفئة الاب (الفئة المجردة) والفئة الابن المشتقة منها، بحيث تكون في صورة مجردة في الفئة المجردة (الفئة الاب) وفي صورة غير مجردة في الفئة غير المجردة (الفئة الابن).

الشكل العام لتعريف الدالة المجردة:

برمجيا يمكننا انشاء الدالة المجردة (Abstract Method) بنفس الطريقة التي يتم بها انشاء الدالة المكتملة (Concrete Methods) على أن تكون مسبوقة بالكلمة المفتاحية abstract وفيما يلي نقدم الشكل العام لإنشاء الدالة المجردة

```
abstract Accessmodifier Abstract_Method_name ();
```

مثال: يوضح هذا المثال عملية انشاء دالة مجردة تحت اسم "eat":

```
abstract  public void eat();
```

مع ملاحظة أن الدالة المجردة يجب أن تعرف داخل فئة المجردة.

مثال : المثال التالي يوضح عملية تعريف فئة مجردة (abstract class) تحت اسم Animal تحتوي على دالة مجردة تحت اسم eat ، كما يوضح هذا المثال عملية تعريف فئة اخري غير مجردة تحت اسم cat تكون موروثة من الفئة المجردة Animal وتحتوي على تطبيق للدالة eat (تكرار للدالة eat في صورة غير مجردة في الفئة الابن) الموجودة في الفئة المجردة Animal.

```
abstract class Animal
{
abstract public void eat();
}
class Cat extends Animal{
    public void eat(){
        System.out.print("cat is eating");
    }
}
```

الوسط البيني Interface:

الوسط البيني interface هو عبارة عن نوع مرجعي مشابه للفئة في امكانية احتوائه على مجموعة من الدوال في صورة مجردة ، حيث يمكن من خلاله تطبيق تقنية التجريد بنسبة 100% وبالتالي فإن الوسط البيني لا يمكن أن يحتوي على دوال غير مجردة، ويتم تنفيذه باستخدام الفئة عن طريق الكلمة المفتاحية implements ، علماً بأن الفئة التي تقوم بتنفيذ (تطبيق) الوسط البيني يجب أن تقوم بتنفيذ كافة الدوال المجردة في الوسط البيني، كما يمكن لهذه الفئة أن تقوم بتطبيق أكثر من وسط بيني واحد في نفس الوقت وهو ما يوفر لنا امكانية تطبيق مفهوم الوراثة المتعددة والتي لا يمكن أن يتم تنفيذها إلا من خلال وجود الوسط البيني.

مثال : المثال التالي يوضح طريقة تعريف وسط بيني يحتوي على دالتين في صورة مجردة، كما يوضح عملية تطبيق الوسط البيني باستخدام فئة تحتوي على نفس الدوال في صورة غيرة مجردة.

```
interface NewInterface{
    public void eat();
    public void sleep();
}
class animal implements NewInterface {
    public void eat(){
    System.out.println("implement eat method");
    }
    public void sleep(){
    System.out.println("implement sleep method");
    }
}
public class Animal2 {
    public static void main(String[] args) {
       animal dog = new animal();
      dog.eat();
      dog.sleep();
    }
}
```

مثال:

```
interface printable{
       void print();
}
class AA implements printable{
public void print(){
       System.out.println("Hello");
}
public static void main(String args[]){
       AA obj = new AA();
       bj.print();
 }
}
```

مخرجات البرنامج Output:

```
Hello
```

الوراثة و الوسط البيني Interface & inheritance:

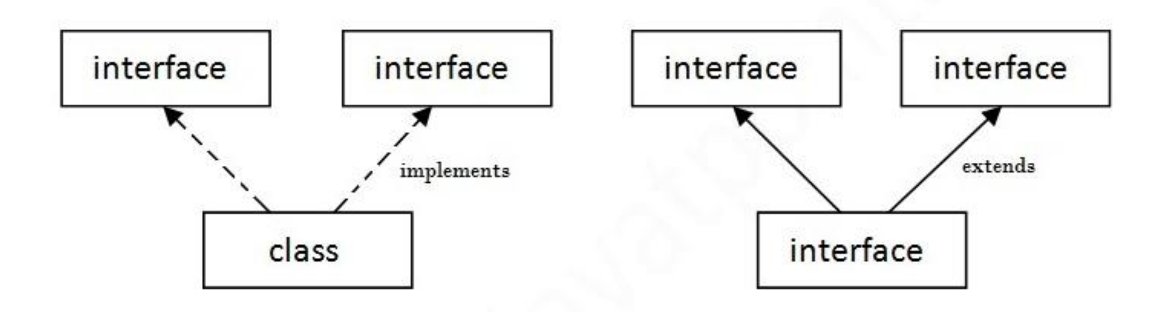
بنفس الطريقة التي يتم بها وراثة فئة لفئة اخرى يمكن لأي وسط بيني أن يرث وسط بيني آخر وذلك باستخدام الكلمة المفتاحية extends، والمثال التالي يوضح طريقة تطبيق مفهوم الوراثة في الوسط البيني.

```
interface NewInterface
{
    public void eat();
    public void sleep();
}

interface NewInterface1 extends NewInterface
{
    public void eat1();
    public void sleep1();
}
interface NewInterface2 extends NewInterface
{
    public void eat2();
    public void sleep2();
}
```

الوراثة المتعددة والوسط البيني:

كما ذكرنا في موضوع الوراثة فان لغة الجافا لا تدعم الوراثة المتعدد للفئات ولتطبيق مفهوم الوراثة المتعدد في لغة الجافا فإنه يجب علينا استخدام الوسط البيني (Interface) حيث أنه إذا قامت الفئة بتنفيذ أكثر من وسط بيني او إذا ورث وسط بيني (ابن) اكثر من وسط بيني (اب) فإن ذلك يسمى بالوراثة المتعددة، ويمكن توضيح فكرة الوراثة المتعددة من خلال الشكل التالي:



مثال: في المثال التالي قمنا بتعريف عدد اثنان وسط بيني (Interface) يحتوي كل منهما على دالة ويتم تطبيقهما باستخدام الفئة (class) المسماة "CC".

```
interface Printable{
      void print();
}
interface Showable{
      void show();
}
class CC implements Printable,Showable{

      public void print(){
      System.out.println("Hello");
}
public void show(){
      System.out.println("Welcome");
}
public static void main(String args[]){
      CC obj = new CC();
      obj.print();
      obj.show();
 }
}
```

مخرجات البرنامج:

```
Hello
Welcome
```